Unlink <modname> Usage : Unlinks module(s) from memory @WCREATE Syntax: Wcreate [opt] or /wX [-s=type] xpos ypos xsiz ysiz fcol bcol [bord] Usage : Initialize and create windows Opts : -? = display help -z = read command lines from stdin -s=type = set screen type for a window on a new screen @XMODE Syntax: Xmode <devname> [params] Usage : Displays or changes theparameters of an SCF type device @COCOPR Syntax: cocopr [<opts>] {<path> [<opts>]} Function: display file in specified format gets defaults from /dd/sys/env.file Options : -c set columns per page -f use form feed for trailer -h=num set number of lines after header -l=num set line length -m=num set left margin -n=num set starting line number and incr -o truncate lines longer than lnlen -p=num set number of lines per page -t=num number of lines in trailer -u do not use title

AUSTRALIAN OS9 NEWSLETTER

-u=title use specfied title -x=num set starting page number -z[=path] read file names from stdin or <path> if given @CONTROL Syntax: control [-e] Usage : Control Panel to set palettes, mouse and keyboard parameters and monitor type for Multi-Vue. Selectable from desk utilities menu as the Control Panel. Opts : -e = execute the environment file @GCLOCK Syntax: gclock Usage : Alarm clock utility for Multi-Vue. Selectable from desk utilities menu as Clock. @GCALC Syntax: gcalc Usage : Graphics calculator utility for Multi-Vue. Selectable form desk utilities menu as Calculator. @GCAL Syntax: gcal Usage : Calendar/Memo book utility for Multi-Vue. Selectable as Calendar from the desk utilities menu. @GPRINT Syntax: gprint Usage : Printer setup utility for Multi-Vue. Lets user graphically

ADDRESSES

Editorial Material:
Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Library Requests:
Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

CONTENTS

Volume 7 **Jan/Feb 1993** **Number 1**

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 7 Number 1

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries
TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet
SUPPORT : Brisbane OS9 Level 2 Users Group.

Here's what's new in the OS9 world.

As you know we are not the only OS9 Usergroup to enjoy the wonders of the OS9 operating system. We are all well aware of the existence of the U.S. Usergroup from which the majority of P.D. material emanates. The present U.S. Usergroup is a revamp of the original group which folded some time ago and it now operates under new rules.

As well as this group, there are other usergroups actively promoting OS9 in their own way, not the least of which is "EUROS9" a usergroup run by Peter Tutelaers in the Netherlands with the support of Burghard Kinzel in Germany. We currently exchange newsletters with this group and are also very grateful for the many P.D. disks which they have forwarded on to us.

The OCN, OS9 Community Network, which communicates through the FIDO network is very active, particularly in the U.S. and Canada.

There is also a relatively new group, I believe, in the U.K. known as the "EUROPEAN OS9 USERGROUP" plus others that we know of, "FOS9" (Japan), "OS9 USERGROUP OF JAPAN" and "EFFO" (Switzerland).

Peter Tutelaers of EUROS9 has contacted us to establish our possible interest in new venture he is proposing to all known OS9 usergroups. This 'venture' is essentially improving communications between the usergroups for exchange of information and public domain software.

Peter proposes to have a few people (two or three) who would collect newsletters and P.D. stuff from nominated usergroups, make copies and then forward on to all usergroups. The details of this scheme are yet to be worked out.

Bob Devries, Don Berrie, Jean-Pierre Jacquet and myself (Gordon Bentzen) discussed this proposal at a very recent newsletter meeting, and we agree that we should participate and support Peter in this 'venture'. After all, each usergroup was set up for the main purpose of exchange of information and for help and assistance to others. This new 'venture' simply extends this concept to an organised International 'usergroup'. Peter has proposed that full details could be worked out at a meeting on May 1st in Chicago USA.

And that leads to the next bit of news. A "CoCoFest" will be held in Chicago over the weekend of May 1st and 2nd? The 'Fest will be run over the Friday night, Saturday and Sunday. In conjunction with that 'Fest, it is proposed to hold a meeting of representatives of OS9 Usergroups around the world. **We have been invited to send a representative to that meeting.**

We have already received an indication that some of the costs involved in sending our representative to the meeting would be covered by the other Usergroups involved. We have not worked out what the costs for our Usergroup would be, but I do expect that some extra funds could be raised by the sale of newsletters and PD-Software from other Usergroups.

Would you as a member of the National OS9 Usergroup be prepared to pay a little more either as an annual subscription, or perhaps a one off donation of say $5, $10 or $15 towards the cost of sending a representative to Chicago?

Remember that the National OS9 Usergroup is operated on a non-profit basis, and that the annual subscription is designed to cover only the production (printing) and mailing costs of the newsletter. This means that we have essentially no carry-over monies from year to year that we could use to help fund this exercise. We have not until now carried any paid advertisements in the Newsletter, and that, after all, is the main source of income for most publications. Perhaps a wider international circulation would make paid ads a proposition, what do you think?

Well, there is something to think about! - Unfortunately we do not have much time to think if we are to have a representative in Chicago in May. We would need to be making definite arrangements by the end of February.

If you think that this is too good an opportunity for the Australian National OS9 Usergroup to miss, a short note in return mail with your pledge of support would be welcome.

Cheers, Gordon.

oooooooooOOOOOOOOOoooooooooo

OS9 Frequently-Asked Questions List
Updated 13 January 1993

This is the fifth revision of the OS9 FAQ list. As in the last edition, addresses will be referenced by number (e.g. [1]) and the address will be listed at the bottom of the file. This (hopefully) will make for easier reading.

In addition, a complete list of the questions answered here is given at the top of the file. Upon compiling the faq, this time, I noticed that there STILL aren't many questions answered here. If you feel there is a question often asked that you don't see answered here, let me know. Supply the answer, also, if you can; otherwise I'll see if I can't track it down.

Again, if there are any additions, corrections, suggestions, comments, flames, or contributions, please respond in kind to me, Russell Hoffman, rh2y+@andrew.cmu.edu

Section One: List of answered questions

Section Two: Q & A

Q: What is OS9?

A: OS9 is a real-time, multiuser, multitasking operating system developed by Microware Systems Corporation. It was originally developed for the 6809 microprocessor, in a joint effort between Microware and Motorola. The original Level I 6809 OS9 OS was capable of addressing 64 kilobytes of memory. The Level II 6809 OS9 took advantage of dynamic address translation hardware, and allowed a mapped address space of one megabyte on most systems, and up to two megabytes on others, most notably the Tandy Color Computer III.

In the 1980's, Microware ported OS9 to the 68000 family of microprocessors, creating OS9/68000. Code is mostly portable from OS9/6809 to OS9/68000 at the high-level-language source code level. Code is compatible within either OS9/6809 or OS9/68000 at the binary level.

OS9/68000 provides synchronization and mutual exclusion primitives in the form of events, which are similar to semaphores. It also allows communication between processes in the form of named and unnamed pipes, as well as shared memory in the form of data modules.

OS9 is modular, allowing new devices to be added to the system simply by writing new device drivers, or if a similar device already exists, by simply creating a new device descriptor. All i/o devices can be treated as files, which unifies the i/o system. In addition, the kernel and all user programs are ROMable. Thus, OS9 can run on any 680x0 based hardware platform from simple diskless embedded control systems to large multiuser minicomputers.

Q: What is OSK?

A: OSK is an abbreviation for OS9/68000. This is probably due to the common abbreviation '68k' for the 68000 microprocessor. Also sometimes called OS9/68k.

Q: Where can I get OS9?

A: Generally the hardware vendor will ship a version of OS9 with the product upon which OS9 is intended to be run. Alternatively, OS9 can be purchased from Microware [1] itself, for certain

hardware platforms. In addition, several software vendors sell customized and enhanced OS9 packages. Ultrascience [2] and ELSOFT AG [3] are examples of such vendors. Note that ELSOFT is in Switzerland.

Q: What machines run OS9?

A: OS9/6809 runs on a variety of platforms, perhaps the most (in)famous being the Tandy Color Computer. Other systems include the SWTPC SCB-69, the Gimix 6809, Smoke Signal Broadcasting's Chieftain 6809, FHL's TC09, the Febe, and many others, most of which are SS-50 bus machines. Note that OS9/6809 is no longer supported by Microware, but many user groups, BBSes, and a handful of FTP sites offer help and maintain software collections for OS9/6809.

OS9/68000 runs on quite a multitude of machines, including a variety of systems from Hazelwood (such as the UniQuad I and II), the Gimix Micro-20, the Atari ST, Commodore Amiga, Apple Macintosh, IMS MM/1, FHL TC-70, and a large number of 680x0-based VME systems, manufactured by such companies as Radstone Technology, Motorola, Heurikon, Inducom, Force, Mizar, and others. Gespac also makes a large number of platforms based on their G-64 and G-96 bus.

Q: Where do I get OS9/68000 for the Macintosh?

A: Ultrascience [2] (A division of Gibbs Laboratories) makes a version of OS9/68000 for the Macintosh. According to their literature, it even allows the Macintosh operating system to run as a process under OS9.

Q: Where do I get OS9/68000 for the Commodore Amiga?

A: Digby Tarvin [4], in Australia, has a port of OS9/68000 for the Amiga, which cost approximately $600 US.

Q: Where do I get OS9/68000 for the Atari ST?

A: Cumana [5] and Dr. Keil [6] offer two different ports of OS9 to the ST.

Q: What is OS-9000?

A: OS-9000 is a portable version of OS9, written primarily in C. It can potentially run on any 68020 or higher 680x0 family member, and any 80386sx or higher 80x86 member. Code is portable across OS-9000 platforms at the source level. Theoretically, OS-9000 can be ported to any modern computer architecture, though 680x0 and 80x86 are the only supported microprocessor families at present.

Q: What software is available for OS9?

A: Nearly any user application can be found either commercially or in the public domain/shareware/freeware. Many word-processor, spreadsheets, databases, and time management software packages are available from a variety of vendors. A list of much of the available commercial software is available from Microware. They publish the "OS9 Sourcebook", a listing of hardware and software vendors who sell both 6809 and 68000 software and hardware. It is advisable to contact the individual companies listed in the Sourcebook and request a recent catalog, as the information in the Sourcebook is a tad outdated. Microware's quarterly magazine Pipelines also carries new product announcements.

Q: Where can I get public-domain/shareware/freeware software for OS9?

A: There are many private bulletin boards around. Hopefully, someone will be publishing a list of all known BBSes which have OS9 software. In addition, there is the Princeton Listserver, which acts as a mailing server that will mail requested software. To begin using the Listserver, send electronic mail to LISTSERV@PUCC.PRINCETON.EDU, with the single line

HELP

in your message. Also, the OS-9 User's Group [7] maintains a library of public domain sofwtare, as well as distributes a newsletter. Finally, there are a few anonymous FTP servers worldwide with OS9 software on them.

Site	Operator	IP address
hermit.cs.wisc.edu	- moved to cabrales.cs.wisc.edu	
cabrales.cs.wisc.edu	Jim Pruyne	128.105.2.70
wuarchive.wustl.edu	Steve Wegert	128.252.135.4
lucy.ifi.unibas.ch	Marc Balmer	131.152.81.1

Don't forget the often overlooked mailserver on cabrales. The e-mail address is os9archive@cabrales.cs.wisc.edu, and a message with "help" as the body will return some help text. This provides a way for those without FTP access to snarf stuff from cabrales via mail.

Cabrales contains mostly OS9/68000 software, including the complete TOP package, many EFFO disks, GCC and G++, (and many other GNU products), ka9q, TeX, and quite a bit of 6809 software.

Wuarchive has mostly 6809 OS9 software; Lucy is meant to be a european duplicate of cabrales.

Q: What is the TOP package?
A: TOP is an acronym for "The OS9 Project". It is a collection of OS9/68000 software developed primarily in Germany. Much of it seems to be an attempt to make OS9 a little more UNIX-like. Many standard unix utilities are provided, as well as a complete UUCP mail implementation, and a more secure password file and login program. Many traditional unix games are also provided. The total package consumes approximately 16 MB of disk space, though much of this is source code.

Q: Are there alternative shells for OS9?
A: Yes, there are. Microware sells mshell, an enhanced shell. In addition, there are several public domain shells available. The most notable of which is the Bourne shell, sh, available in the TOP package (OS9/68000). It supports aliasing, command-line editing, history, environment variable replacement, shell scripting, the `command` operator (which uses the output of the command as arguments to the called program), and a startup file. A PD version of ksh is available on cabrales.

For OS9/6809, there is Shell+ and of course if you have a Color Computer, there is always Gshell, a graphical shell.

Q: Can one read/write MS-DOS format disks under OS9?
A: Yes, there are several public-domain and commercially avaliable utilities to accomplish this task, for both OS9/6809 and OSK. One of the more interesting is the MSFM file manager which appears in _OS9_Insights_, a book by Peter Dibble, available through Microware. MSFM is an actual file manager, which allows you to mount an MS-DOS floppy as part of the OS9 file system.

Q: What sorts of communications software is available?
A: Many public domain utilities, available from your local BBS, include terminal emulators and file transfer utilities (such as xmodem, ymodem, zmodem, and kermit protocols.) Sterm, a non-commercial package, also supports Compuserve B+ protocol. In addition, many software vendors sell various equivalent packages. C-kermit is available in source and executable form for OS9/68000 on cabrales.

Also, Microware sells the NFM Network File Manager, which is a local-area networking protocol for small networks of strictly OS9 based computers. NFM runs on virtually any network interface, including direct serial links, ARCnet, Ethernet, and others.

Microware also sells the ISP, or Internet Support Package, which is a relatively complete TCP/IP package, including telnet client and server apllications, and FTP client and server. It also provides a C BSD 4.2 compatible socket library. Closely related is the ESP, or Ethernet Support Package. This is similar to ISP, but is for particular Ethernet boards. Current word from Microware says that the ESP is now obsolete, and has been replaced by a preconfigured version of the ISP.ISP supports Ethernet and SLIP, although there is no current SLIP driver supported by Microware.

Microware also sells NFS, or Network File System, for OS9/68000. This allows an OS9 system to share files in a heterogeneous environment (i.e. not all the machines on the network run OS9.) NFS requires ISP or ESP.

Finally, there is a port of the Phil Karn ka9q internet software package, which supports a single-user interface to TCP/IP. It includes a telnet client, an FTP client and server, and SMTP. Source and executables may be found on cabrales. Note that the executables on cabrales have a bug in the FTP server which causes it to bus trap occasionally. Hopefully someone will take the time to find this and correct it.

Q: What about usenet and news?
A: Several ports of UUCP software are available for both os9/6809 and os9/68k. A port of C news and Rn are available on cabrales. TOP has ported Notes, which maintains Notesfiles. There is a program which will transfer between Notesfiles and netnews. The TOP package in its entirety may be found on cabrales.

Rick Adams' UUCP port for the Color Computer may be found on wuarchive, as well as on Delphi and Compuserve.

Q: Is gcc available for OS9?
A: gcc and g++ are available for OS9/68000, both in OS9 executable form and cross-compiler form. Versions 1.37, 1.39 and 1.40 were ported to OS9/68000 primarily via the work of Stephan Paschedag. Source and binaries are available on cabrales.cs.wisc.edu via anonymous FTP. The 1.40 version supports 68040 optimizations.

Q: Can I run X11 on OS9?
A: Yes. Microware sells a port of X11R4 (client

and server plus optional Motif) , as well as do Eltek Electronik GmbH [8]

Q: What other graphics alternatives are there?
A: Several other organizations have various graphics packages for OS9. Reccoware Systems [9] has a port of the Bellcore MGR window manager. Gespac [10] produces G-windows, a portable windowing package which has device windows and a very Motif-looking interface. For the MM/1, Interactive Media Systems [11] is producing K-windows, window manager similar to Multi-Vue, the OS9 window package for the Tandy Color Computer III. Microware also sells RAVE, the Real-Time Audio Video Environment.

Q: What is a Real Time system?
A: A real-time system is any system whose correctness depends not only on the correctness of the applied algorithms, but also in the timing of the execution of those algorithms. Refer to the netnews comp.realtime newsgroup for more information.

Q: Does OS9 support multiple threads within a program?
A: No, not directly like Mac does, but through the use of user installed periodic interrupts or alarms, a user program can support it's own threads. Consult a good operating systems book for more details.

Section Three: Addresses, by reference number

[1]——————————
Microware Systems Corporation
1900 N.W. 114th Street Des Moines, Iowa 50322
Phone: (515) 224-1929 Fax: +1 (515) 224-1352

Microware Systems Corporation
Western Regional Office
2041 Mission College Boulevard Santa Clara,
California 95054
Phone +1 (408) 980-0201 Fax +1 (408) 980-1671

Northeastern Regional Office
One Crank Rd Hampton Falls, NH 03844
(603)929-4107 (603)929-4233 fax

Southeastern Regional Office
P.O. Box 510358 Melbourne Beach, FL 32951-0358
(407)725-2840 (407)725-2487 fax

Microware Systems (UK) Limited
Leylands Farm, Nobs Crook
Colden Common Winchester, Hants.
England, SO21 1TH
Phone: +44 703 601990 Fax: +44 703 601991

Microware Systems K.K.
17-3, Sotokanda 2-Chome
Chiyoda-Ku Tokyo 101, Japan
Phone: +81 3-3257-9000 Fax: +81 3-3257-9200

Microware Systems France
Chateau de la Saurine
Pont de Bayeux 13590
Meyreuil France
Phone: +33 42 58 63 00 Fax: +33 42 58 62 28

[2]——————————
Ultrascience Box 847 Wheeling,
Illinois 60090 USA
(708)-808-9060 FAX: (708)-808-9061

[3]——————————
ELSOFT AG Zelgweg
12 CH-5405 Baden-Daettwil
Tel. +41 56 83 33 77 Fax. +41 56 83 30 20

[4]——————————
Digby Tarvin, Technical Director
Tesseract Pty. Ltd
Computer Consultants
53 George St.
Redfern,
New South Wales Australia, 2016
Fax: 011-61-2-698-8881

Email: digbyt@extro.ucc.su.oz.au

[5]——————————
Cumana Ltd.
The Pines Trading Estate
Broad Street
Guildford Surrey England, GU3 3BH
Phone: +44 483 503121 Fax: +44 483 503326

[6]——————————
Dr. R. Keil GmbH
Gerhart-Hauptmann-Str.
30 D-6915 Dossenheim
Tel. +49 6221 86 20 91 Fax. +49 6221 86 19 54

[7]——————————
OS-9 User's Group
PO Box 465 Goshen,
Indiana USA 46526-0465

[8]——————————
Eltek Electronik GmbH
Galileo-Galilei-Strasse 11
D-6500 Mainz 42
Germany
Phone: (6131) 588-0 fax: (6131) 588-199

[9]——————————
Reccoware Systems
Wolfgang Ocker
Lochhauser Strasse 35a
D-8039 Puchheim
Voice: +49 89 80 77 02 Fax: 49 89 80 29 67

[10]——————————
To contact Gespac,
call toll-free 1-800-4GESPAC

[11]——————————
Interactive Media Systems
238 Catawba Avenue,
Davidson,
North Carolina USA 28036
Phone: 704/892-6233

or

IMS SALES
1840 Biltmore Street NW
Suite 10
Washington DC USA 20009
Phone:(202) 232-4246

oooooooooooOOOOOOOOOOooooooooooo

Saving window status in Basic09
by Bob Devries

One of my pet hates with programmes written for OS9 (and indeed any computer), is programmes which having created a special screen type for running in, proceed to quit WITHOUT RESTORING the screen that I was running.

The fault lies squarely with the (lazy) programmer, because it is mostly very easy to save the current screen settings, and restore them after the programme has finished. I guess the place to start is to save the current settings from what the techos call the PD.OPTS section. The information from the PD.OPTS can be easily saved in a 32 byte buffer (make sure it is in the main routine of your programme). Here is a Basic09 sample:-

```
TYPE registers=cc,a,b,dp:byte; x,y,u:integer
DIM regs:registers
DIM getstt,setstt:byte
DIM opt_buff(32):byte

getstt=$8D \ (* Get Status system call
setstt=$8E \ (* Set Status system call

regs.a=0 \ (* use STDIN all others are automatic
regs.b=0 \ (* GS_OPT call
regs.x=ADDR(opt_buff) \ (* address of 32 byte buffer
RUN syscall(getstt,regs) \ (* do the job
```

That piece of code is broken up into the DIMension statements, including the TYPE complex variable for the CPU registers, setting the various variables to their values (see comments), and the actual code for the operation. You only need to change the PD.OPTS of ONE of the standard paths, all the others merely mirror STDIN, so what you change in one, is also changes in all the others.

The code to reset the PD.OPTS is very simple:-

```
regs.a=0 \ (* STDIN again
regs.b=0 \ (* SS_OPT call
regs.x=ADDR(opt_buff) \ (* address of 32 byte buffer of previous
RUN syscall(setstt,regs) \ (* Set Status call this time.
```

Now you may quit.

OK, but I'm not finished yet!

We still need to deal with changes to the screen, like screen type, palette colours, and foreground, background, and border colours.

Now, there are (at least) two ways to do this. I have used both successfully, so I will show you what I have done. The first method assumes plenty of memory, because it involves creating a new screen OVER THE TOP OF the existing screen. This is a bit more involved for future pieces of code as you will see. Here is the code:- (I'm assuming previously DIMmed variables for clarity.)

```
(* first open a path to a new descriptor
OPEN #wpath,"/w":UPDATE \ (* UPDATE essential here
RUN gfx2(#wpath,"DWSET",7,0,0,80,24,0,1,2) \ (set up type 7 graphics screen
RUN gfx2(#wpath,"SELECT") \ (* make the new screen visible
RUN gfx2(#wpath,"FONT",200,1) \ (* select font to use
```

Now, that piece of code assumes 1. that you have the descriptor '/w' in memory (from your boot), and 2. that you have merged sys/stdfonts (although this can be done within the programme).

One niggly problem with this approach is that ALL print statements and gfx2 commands MUST quote the #wpath, else the printing will go to the (invisible) window underneath.

To reselect the original window, here's what you do:-

```
RUN gfx2(0,"SELECT") \ (* select the old screen
RUN gfx2(#wpath,"DWEND") \ (* kill off the old screen
```

OK, so much for the first method. Here's another way:-

```
DIM pals(16):BYTE \ (* Palette reg buffer
DIM sels(3)BYTE \ (* Fore, background and border regs buffer
DIM sctyp:BYTE \ (* screen type code 1..8
DIM screen_x:INTEGER
DIM screen_y:INTEGER
TYPE registers=cc,a,b,dp:BYTE; x,y,u:INTEGER
DIM regs:registers
DIM getstt,setstt:BYTE

getstt=$8D
setstt=$8E
regs.a=0 \ (* stdin path
regs.b=$91 \ (*SS.Palet
regs.x=ADDR(pals) \ (* address of palette buffer
RUN syscall(getstt,regs) \ (* do Get Status call
```

You now have the palette registers of the old screen.

```
regs.a=0 \ (* stdin path
regs.b=$96 \ (* SS.FBRgs
regs.x=ADDR(sels) \ (* buffer for colour registers
RUN syscall(getstt,regs) \ (* do Get Status call
```

Now you have the colour registers.

```
regs.a=0 \ (* stdin...
regs.b=$93 \ (* SS.ScTyp
RUN syscall(getstt,regs) \ (* do Get Status call
sctyp=regs.a \ (* save screen type returned in A reg
```

Now you have the screen type.

```
regs.a=0 \ (* stdin...
regs.b=$26 \ (* SS.ScSiz
RUN syscall(getstt,regs) \ (* do Get Status call
screen_x=regs.x \ (* X size in x reg
screen_y=regs.y \ (* Y size in y reg
```

Now you have the screen size. This info is probably all you need. I know that things like working area can be different, but most of the info is saved, though you could probably take this to all sorts of depths. Now you can kill off the old screen, and restart a new one.

```
RUN gfx2(0,"DWEND") \ (* Kill old screen
RUN gfx2("DWSET",7,0,0,80,24,0,1,2) \ (* set up your screen
RUN gfx2("SELECT") \ (* make it visible
RUN gfx2("FONT",200,1) \ (* select a font
```

Now you can use the screen to your heart's content, and you don't have to worry about a path to the screen. To restore, you must reverse the previous procedure, although, not in as many steps.

```
RUN gfx2(0,"DWEND")
regs.a=0 \ (* stdin...
regs.b=$97 \ (* SS.DFPal
regs.x=ADDR(pals) \ (* palette regs
RUN syscall(setstt,regs) \ (* do Set Status call
RUN                                    gfx2("DWS
ET",sctyp,0,0
,screen_x,screen_y,pals(1),pals(2),pals(3))
RUN gfx2("SELECT") \ (* make it visible
```

Now you are back to where you started. If you had a need to change things like the screen pause etc., you should have also used the routine at the start of this article. One point I must stress again, is that you MUST use variables declared in the main part of your programme (as in main() in C programming) because otherwise your variable contents will be lost.

I hope that this will help programmers (and experimenters) to programme more neatly, and perhaps I have been able to impart some of my experiences to someone. If so, great.

Regards, Bob Devries

oooooooooooOOOOOOOOOOooooooooooo

Windows for Basic09 - another way

by Bob Devries, programme by Gene Krenciglowa

After I wrote the article on Basic09 and windows for this issue, I came across a sample programme whilst perusing messages on the OS9 echo on FIDONET. It just shows that I can't think of

everything, and that there is always some other way of doing the same thing. In this case, Gene's way has some distinct advantages, and is a great example of the use of the I$DUP system call from within Basic09. Aint science wonderful! Thanks, Gene!

```
PROCEDURE dwsel
TYPE registers=cc,a,b,dp:BYTE; x,y,u:INTEGER
DIM regs:registers
DIM win1,Dup,outstd,outdup,wlast:BYTE
outstd=1 \Dup=$82
RUN showscrn(outstd)
regs.a=outstd \RUN syscall(Dup,regs) \outdup=regs.a
PRINT "outdup  "; outdup
CLOSE #outstd \(* free up path 1
OPEN #win1,"/w":UPDATE
RUN gfx2(win1,"DWSET",1,0,0,40,24,0,2,1)
RUN gfx2("SELECT") \RUN delay
RUN showscrn(win1)
RUN gfx2(outdup,"SELECT")  \(*  ***  the key line -
before CLOSE #win1
RUN delay
CLOSE #win1 \(* free up path 1
regs.a=outdup \RUN syscall(Dup,regs) \wlast=regs.a
CLOSE #outdup
RUN showscrn(wlast)
END

PROCEDURE showscrn
PARAM wn:BYTE
DIM zz:STRING[1]
PRINT "path number = "; wn; "  press any key"
GET #wn,zz
END

PROCEDURE delay
DIM i:INTEGER
RUN gfx2("BELL") \ FOR i=1 TO 10000 \NEXT i
END
```

By the way, Gene is from Canada. The tagline on his message reads: DISCUS BBS, Hull Quebec,Canada (819)771-3792 (1:163/519).

Bob Devries

oooooooooOOOOOOOOOooooooooo

A C Tutorial
Chapter 12 - Dynamic Allocation

WHAT IS DYNAMIC ALLOCATION?

Dynamic allocation is very intimidating to a person the first time he comes across it, but that need not be. Simply relax and read this chapter carefully and you will have a good grounding in a very valuable programming resource. All of the variables in every program up to this point have been static variables as far as we are concerned. (Actually, some of them have been "automatic" and were dynamically allocated for you by the system, but it was transparent to you). In this chapter, we will study some dynamically allocated variables. They are simply variables that do not exist when the program is loaded, but are created dynamically as they are needed. It is possible, using these techniques, to create as many variables as needed, use them, and deallocate their space for use by other variables. As usual, the best teacher is an example, so load and display the program named DYNLIST.C.

We begin by defining a named structure "animal" with a few fields pertaining to dogs. We do not define any variables of this type, only three pointers. If you search through the remainder of the program, you will find no variables defined so we have nothing to store data in. All we have to work with are three pointers, each of which point to the defined structure. In order to do anything, we need some variables, so we will create some dynamically.

DYNAMIC VARIABLE CREATION

The first program statement, which assigns something to the pointer "pet1" will create a dynamic structure containing three variables. The heart of the statement is the "malloc" function buried in the middle of the statement. This is a "memory allocate" function that needs the other things to completely define it. The "malloc" function, by default, will allocate a piece of memory on a "heap" that is "n" characters in length and will be of type character. The "n" must be specified as the only argument to the function. We will discuss "n" shortly, but first we need to define a "heap".

WHAT IS A HEAP?

Every compiler has a set of limitations on it as to how big the executable file can be, how many variables can be used, how long the source file can be, etc. One limitation placed on users by many compilers for the IBM-PC and compatibles is a limit of 64K for the executable code. This is because

the IBM-PC uses a microprocessor with a 64K segment size, and it requires special calls to use data outside of a single segment. In order to keep the program small and efficient, these calls are not used, and the program is limited but still adequate for most programs.

A heap is an area outside of this 64K boundary which can be accessed by the program to store data and variables. The data and variables are put on the "heap" by the system as calls to "malloc" are made. The system keeps track of where the data is stored. Data and variables can be deallocated as desired leading to holes in the heap. The system knows where the holes are and will use them for additional data storage as more "malloc" calls are made. The structure of the heap is therefore a very dynamic entity, changing constantly.

MORE ABOUT SEGMENTS

Some of the more expensive compilers give the user a choice of memory models to use. Examples are Lattice and Microsoft, which allow the programmer a choice of using a model with a 64K limitation on program size but more efficient running, or using a model with a 640K limitation and requiring longer address calls leading to less efficient addressing. Using the larger address space requires inter segment addressing resulting in the slightly slower running time. The time is probably insignificant in most programs, but there are other considerations.

If a program uses no more than 64K bytes for the total of its code and memory and if it doesn't use a stack, it can be made into a .COM file. Since a .COM file is already in a memory image format, it can be loaded very quickly whereas a file in a .EXE format must have its addresses relocated as it is loaded. Therefore a small memory model can generate a program that loads faster than one generated with a larger memory model. Don't let this worry you, it is a fine point that few programmers worry about.

Using dynamic allocation, it is possible to store the data on the "heap" and that may be enough to allow you to use the small memory model. Of course, you wouldn't store local variables such as counters and indexes on the heap, only very large arrays or structures. Even more important than the need to stay within the small memory model is the need to stay within the computer. If you had a program that used several large data storage areas, but not at the same time, you could load one block storing it dynamically, then get rid of it and reuse the space for the next large block of data. Dynamically storing each block of data in succession, and using the same storage for each block may allow you to run your entire program in the computer without breaking it up into smaller programs.

BACK TO THE "MALLOC" FUNCTION

Hopefully the above description of the "heap" and the overall plan for dynamic allocation helped you to understand what we are doing with the "malloc" function. It simply asks the system for a block of memory of the size specified, and gets the block with the pointer pointing to the first element of the block. The only argument in the parentheses is the size of the block desired and in our present case, we desire a block that will hold one of the structures we defined at the beginning of the program. The "sizeof" is a new function, new to us at least, that returns the size in bytes of the argument within its parentheses. It therefore, returns the size of the structure named animal, in bytes, and that number is sent to the system with the "malloc" call. At the completion of that call, we have a block on the heap allocated to us, with petl pointing to the first byte of the block.

WHAT IS A CAST?

We still have a funny looking construct at the beginning of the "malloc" function call. That is called a "cast". The "malloc" function returns a block with the pointer pointing to it being a pointer of type "char" by default. Many times, if not most, you do not want a pointer to a "char" type variable, but to some other type. You can define the pointer type with the construct given on the example line. In this case we want the pointer to point to a structure of type "animal", so we tell the compiler with this strange looking construct. Even if you omit the cast, most compilers will return a pointer correctly, give you a warning, and go on to produce a working program. It is better programming practice to provide the compiler with the cast to prevent getting the warning message.

USING THE DYNAMICALLY ALLOCATED MEMORY BLOCK

If you remember our studies of structures and pointers, you will recall that if we have a structure with a pointer pointing to it, we can access any of the variables within the structure. In the next three lines of the program, we assign some silly data to the structure for illustration. It should come as no surprise to you that these assignment statements look just like assignments to statically defined variables.

In the next statement, we assign the value of "pet1" to "pet2" also. This creates no new data, we simply have two pointers to the same object. Since "pet2" is pointing to the structure we created above, "pet1" can be reused to get another dynamically allocated structure which is just what we do next. Keep in mind that "pet2" could have just as easily been used for the new allocation. The new structure is filled with silly data for illustration.

Finally, we allocate another block on the heap using the pointer "pet3", and fill its block with illustrative data.

Printing the data out should pose no problem to you since there is nothing new in the three print statements. It is left for you to study.

GETTING RID OF THE DYNAMICALLY ALLOCATED DATA

Another new function is used to get rid of the data and free up the space on the heap for reuse, the function "free". To use it, you simply call it with the pointer to the block as the only argument, and the block is deallocated.

In order to illustrate another aspect of the dynamic allocation and deallocation of data, an additional step is included in the program on your monitor. The pointer "pet1" is assigned the value of "pet3". In doing this, the block that "pet1" was pointing to is effectively lost since there is no pointer that is now pointing to that block. It can therefore never again be referred to, changed, or disposed of. That memory, which is a block on the heap, is wasted from this point on. This is not something that you would ever purposely do in a program. It is only done here for illustration.

The first "free" function call removes the block of data that "pet1" and "pet3" were pointing to, and the second "free" call removes the block of data that "pet2" was pointing to. We therefore have lost access to all of our data generated earlier. There is still one block of data that is on the heap but there is no pointer to it since we lost the address to it. Trying to "free" the data pointed to by "pet1" would result in an error because it has already been "freed" by the use of "pet3". There is no need to worry, when we return to DOS, the entire heap will be disposed of with no regard to what we have put on it. The point does need to made that losing a pointer to a block of the heap, forever removes that block of data storage from our program and we may need that storage later.

Compile and run the program to see if it does what you think it should do based on this discussion.

THAT WAS A LOT OF DISCUSSION

It took nearly four pages to get through the discussion of the last program but it was time well spent. It should be somewhat exciting to you to know that there is nothing else to learn about dynamic allocation, the last four pages covered it all. Of course, there is a lot to learn about the technique of using dynamic allocation, and for that reason, there are two more files to study. But the fact remains, there is nothing more to learn about dynamic allocation than what was given so far in this chapter.

AN ARRAY OF POINTERS

Load and display the file BIGDYNL.C for another example of dynamic allocation. This program is very similar to the last one since we use the same structure, but this time we define an array of pointers to illustrate the means by which you could build a large database using an array of pointers rather than a single pointer to each element. To keep it simple we define 12 elements in the array and another working pointer named "point".

The "*pet[12]" is new to you so a few words would be in order. What we have defined is an array of 12 pointers, the first being "pet[0]", and the last "pet[11]". Actually, since an array is itself a pointer, the name "pet" by itself is a pointer to a pointer. This is valid in C, and in fact you can go farther if needed but you will get quickly confused. I know of no limit as to how many levels of pointing are possible, so a definition such as "int ****pt" is legal as a pointer to a pointer to a pointer to a pointer to an integer type variable, if I counted right. Such usage is discouraged until you gain considerable experience.

Now that we have 12 pointers which can be used like any other pointer, it is a simple matter to write a loop to allocate a data block dynamically for each and to fill the respective fields with any data desirable. In this case, the fields are filled with simple data for illustrative purposes, but we could be reading in a database, readings from some test equipment, or any other source of data. A few fields are randomly picked to receive other data to illustrate that simple assignments can be used, and the data is printed out to the monitor. The pointer "point" is used in the printout loop only to serve as an illustration, the data could have been easily printed using the "pet[n]" means of definition. Finally, all 12 blocks of data are freed before terminating the program.

Compile and run this program to aid in

understanding this technique. As stated earlier, there was nothing new here about dynamic allocation, only about an array of pointers. A LINKED LIST We finally come to the grandaddy of all programming techniques as far as being intimidating. Load the program DYNLINK.C for an example of a dynamically allocated linked list. It sounds terrible, but after a little time spent with it, you will see that it is simply another programming technique made up of simple components that can be a powerful tool.

In order to set your mind at ease, consider the linked list you used when you were a child. Your sister gave you your birthday present, and when you opened it, you found a note that said, "Look in the hall closet." You went to the hall closet, and found another note that said, "Look behind the TV set." Behind the TV you found another note that said, "Look under the coffee pot." You continued this search, and finally you found your pair of socks under the dogs feeding dish. What you actually did was to execute a linked list, the starting point being the wrapped present and the ending point being under the dogs feeding dish. The list ended at the dogs feeding dish since there were no more notes.

In the program DYNLINK.C, we will be doing the same thing as your sister forced you to do. We will however, do it much faster and we will leave a little pile of data at each of the intermediate points along the way. We will also have the capability to return to the beginning and retraverse the entire list again and again if we so desire.

THE DATA DEFINITIONS

This program starts similarly to the last two with the addition of the definition of a constant to be used later. The structure is nearly the same as that used in the last two programs except for the addition of another field within the structure, the pointer. This pointer is a pointer to another structure of this same type and will be used to point to the next structure in order. To continue the above analogy, this pointer will point to the next note, which in turn will contain a pointer to the next note after that.

We define three pointers to this structure for use in the program, and one integer to be used as a counter, and we are ready to begin using the defined structure for whatever purpose we desire. In this case, we will once again generate nonsense data for illustrative purposes.

THE FIRST FIELD

Using the "malloc" function, we request a block of storage on the "heap" and fill it with data. The additional field in this example, the pointer, is assigned the value of NULL, which is only used to indicate that this is the end of the list. We will leave the pointer "start" at this structure, so that it will always point to the first structure of the list. We also assign "prior" the value of "start" for reasons we will see soon. Keep in mind that the end points of a linked list will always have to be handled differently than those in the middle of a list. We have a single element of our list now and it is filled with representative data.

FILLING ADDITIONAL STRUCTURES

The next group of assignments and control statements are included within a "for" loop so we can build our list fast once it is defined. We will go through the loop a number of times equal to the constant "RECORDS" defined at the beginning of our program. Each time through, we allocate memory, fill the first three fields with nonsense, and fill the pointers. The pointer in the last record is given the address of this new record because the "prior" pointer is pointing to the prior record. Thus "prior->next" is given the address of the new record we have just filled. The pointer in the new record is assigned the value "NULL", and the pointer "prior" is given the address of this new record because the next time we create a record, this one will be the prior one at that time. That may sound confusing but it really does make sense if you spend some time studying it.

When we have gone through the "for" loop 6 times, we will have a list of 7 structures including the one we generated prior to the loop. The list will have the following characteristics.

1. "start" points to the first structure in the list.
2. Each structure contains a pointer to the next structure.
3. The last structure has a pointer that points to NULL and can be used to detect the end.

```
start->struct1 This diagram should aid in
    name    understanding the structure of
    breed   the data at this point.
    age
    point->struct2
        name
        breed
        age
        point->struct3
            name
            breed
            age
```

```
point-> . . . . struct7
                name
                breed
                age
                point->NULL
```

It should be clear to you, if you understand the above structure, that it is not possible to simply jump into the middle of the structure and change a few values. The only way to get to the third structure is by starting at the beginning and working your way down through the structure one record at a time. Although this may seem like a large price to pay for the convenience of putting so much data outside of the program area, it is actually a very good way to store some kinds of data.

A word processor would be a good application for this type of data structure because you would never need to have random access to the data. In actual practice, this is the basic type of storage used for the text in a word processor with one line of text per record. Actually, a program with any degree of sophistication would use a doubly linked list. This would be a list with two pointers per record, one pointing down to the next record, and the other pointing up to the record just prior to the one in question. Using this kind of a record structure would allow traversing the data in either direction.

PRINTING THE DATA OUT

To print the data out, a similar method is used as that used to generate the data. The pointers are initialized and are then used to go from record to record reading and displaying each record one at a time. Printing is terminated when the NULL on the last record is found, so the program doesn't even need to know how many records are in the list. Finally, the entire list is deleted to make room in memory for any additional data that may be needed, in this case, none. Care must be taken to assure that the last record is not deleted before the NULL is checked. Once the data is gone, it is impossible to know if you are finished yet.

MORE ABOUT DYNAMIC ALLOCATION AND LINKED LISTS

It is not difficult, and it is not trivial, to add elements into the middle of a linked lists. It is necessary to create the new record, fill it with data, and point its pointer to the record it is desired to precede. If the new record is to be installed between the 3rd and 4th, for example, it is necessary for the new record to point to the 4th record, and the pointer in the 3rd record must point to the new one. Adding a new record to the beginning or end of a list are each special cases. Consider what must be done to add a new record in a doubly linked list.

Entire books are written describing different types of linked lists and how to use them, so no further detail will be given. The amount of detail given should be sufficient for a beginning understanding of C and its capabilities.

ANOTHER NEW FUNCTION - CALLOC

One more function must be mentioned, the "calloc" function. This function allocates a block of memory and clears it to all zeros which may be useful in some circumstances. It is similar to "malloc" and will be left as an exercise for you to read about and use "calloc" if you desire.

PROGRAMMING EXERCISES

1. Rewrite the example program STRUCT1.C from chapter 11 to dynamically allocate the two structures.
2. Rewrite the example program STRUCT2.C from chapter 11 to dynamically allocate the 12 structures.

oooooooooooOOOOOOOOOOooooooooooo

AUSTRALIAN - NATIONAL OS9 USER GROUP - MEMBERS as at 01/31/93 mm/dd/yy

AMBROSI	G. A.	172 OGILVIE STREET	ESSENDON	VIC 3040
BAILEY	Eric	61 WINCHESTER STREET	MOONEE PONDS	VIC 3039
BARBELER	Keith J	37 KUNDE STREET	LOGANHOLME	QLD 4129
BARKER	Robert	P.O. BOX 711	LIVERPOOL	NSW 2170
BEAMISH	T	1 MOUNTAIN HWY	WANTIRNA	VIC 3152
BEASLEY	Drew	7 ESKDALE STREET	HOLMVIEW	QLD 4207
BENTZEN	Gordon	8 ODIN STREET	SUNNYBANK	QLD 4109
BERRIE	Don	25 IRWIN TERRACE	OXLEY	QLD 4075

BLAZEJEWSKI	Stan	51 EVAN STREET	PARKDALE	VIC 3195
BOARDMAN	William	10 ELTHAM AVENUE	PORT LINCOLN	SA 5606
BYE	Graham	9 AIRLIE BANK RD	MORWELL	VIC 3840
CALE	David	23 HORNSEY ROAD	FLOREAT PARK	WA 6314
COSSAR	M. E.	12 RAKEIORA GROVE	PETONE 6303	NEW ZEALAND
CUNNINGHAM	Eric	7 NUTHATCH STREET	INALA	QLD 4077
DALZELL	Robbie	31 NEDLAND CRES.	PT.NOARLUNGA STH	SA 5167
DEVRIES	Bob	21 VIRGO STREET	INALA	QLD 4077
DONALDSON	Andrew	5 THE GLADES	DONCASTER	VIC 3108
DONGES	Geoff	P.O.BOX 326	KIPPAX	ACT 2615
DUNN	Wally	2/10 TARANTO RD	MARSFIELD	NSW 2122
EATON	David	20 GREGSON PLACE	CURTIN	ACT 2605
EDWARDS	Peter	40 DAVISON STREET	MITCHAM	VIC 3132
EVANS	John	80 OSBURN DRIVE	MACGREGOR	ACT 2615
GALL	Brian D	PO Box 131	COORANBONG	NSW 2265
GODFREY	Stephen	HEINEMANN ROAD	REDLAND BAY	QLD 4165
HARBECKE	Peter	18 MACHENZIE STREET	MANLY WEST	QLD 4179
HARRY	Peter	47 GLENWOOD DRIVE	MORAYFIELD	QLD 4506
HASKELL	Ray	34 LEITCHS RD	ALBANY CREEK	QLD 4035
HOCKLEY	Terry	RM G12, BUILDING 13	UNI OF SA 5095	POORAKA
HOLDEN	Rod	53 HAIG ROAD	LOGANLEA	QLD 4131
HUTCHINSON	Simon	10 ASCOT COURT	NTH DANDENONG	VIC 3175
IKIN	John C	42 SPRUCE DRIVE	ROWVILLE	VIC 3178
JACQUET	J.P.	27 HAMPTON STREET.	DURACK	QLD 4077
JOHNSON	FRASER	35 ROBSON AVE	GOROKAN	NSW 2263
KINZEL	Burghard	LEIPZIGER RING 22A	5042 ERFTSTADT	GERMANY
LARAWAY	Terry	41 N.W. DONCEE DRIVE	BREMERTON U.S.A.	WA 98310
LIDGARD	Ron	P.O.Box 3002	VICTORIA POINT W	QLD 4165
MACKAY	Rob	7 HARBURG DRIVE	BEENLEIGH FOREST	QLD 4207
MacKENZIE	Greg	346 COOK ROAD	HMAS CERBERUS	VIC 3920
MANZI	Sam	73 CURTIN AVE	LALOR	VIC 3075
MARTIN	TED	7 NILE AVENUE	SANDY BAY	TAS 7005
McGIVERN	Jim	39 BANK STREET	MEADOWBANK	NSW 2114
McGRATH	A. John	93 LEMON GUMS DRIVE	TAMWORTH	NSW 2340
McLINTOCK	George	7 LOGAN STREET	NARRABUNDAH	ACT 2604
McMASTER	Brad	P.O. BOX 1190	CROWS NEST	NSW 2065
McNABB	John	PO BOX 109	BORONIA	VIC 3155
MORTON	David	c/o P.O.BOX 195	CONDOBOLIN	NSW 2877
MURPHY	Kevin	108 ADENEY AVE	KEW	VIC 3101
MURRELLS	Alan	5 GOULBURN AVE	CORIO	VIC 3214
PITRE	Boisy G	PO BOX 523	WAUKEE U.S.A.	IA 50263
PRIMAVERA	Camillo	29 RICHARD STREET	MARYBOROUGH	QLD 4650
REMIN	Fred	11 CORCORAN CRES	CANUNGRA	QLD 4275
SINGER	Maurice	217 PRESTON ROAD	WYNNUM WEST	QLD 4178
SKEBE	Jeff	23 NORMA ROAD	PALM BEACH	NSW 2108
STEMAN	John	P.O.BOX 680	WINDSOR	NSW 2755
STEVENS	Darren	16 HEWITT ST	COLAC	VIC 3250
SWINSCOE	Robin	17 MELALEUCA STREET	SLADE POINT	QLD 4740
TAYLOR	Ray	18 CONIFER ST	BROWNS PLAINS	QLD 4118
UNSWORTH	Rob	20 SALISBURY ROAD	IPSWICH	QLD 4305
WINTER	Neville	153 LASCELLES STREET	BRIGHTON	QLD 4017
Total Members	59			

oooooooooooOOOOOOOOOOooooooooooo

NOW AVAILABLE: A Queensland based BBS for OS9 Users. Phone number is (07) 2009870, up to 2400 baud, and it currently runs from 9.00 PM until 11.00 PM. (AEST) Give us a call!